1-1-2013

Suitability assessment framework of agent-based software architectures

Ghassan Beydoun
University of Wollongong, beydoun@uow.edu.au

Graham Low
University of New South Wales, g.low@unsw.edu.au

Paul Bogg
University Of New South Wales

Suitability assessment framework of agent-based software architectures

Abstract

Context: A common distributed intelligent system architecture is Multi Agent Systems (MASs). Creating systems with this architecture has been recently supported by Agent Oriented Software Engineering (AOSE) methodologies. But two questions remain: how do we determine the suitability of a MAS implementation for a particular problem? And can this be determined without AOSE expertise? Objective: Given the relatively small number of software engineers that are AOSE experts, many problems that could be better solved with a MAS system are solved using more commonly known but not necessarily as suitable development approaches (e.g. object-oriented). The paper aims to empower software engineers, who are not necessarily AOSE experts, in deciding whether or not they should advocate the use of an MAS technology for a given project. Method: The paper will construct a systematic framework to identify key criteria in a problem requirement definition to assess the suitability of a MAS solution. The criteria are first identified using an iterative process. The features are initially identified from MAS implementations, and then validated against related work. This is followed by a statistical analysis of 25 problems that characterise agent-oriented solutions previously developed to group features into key criteria. Results: Key criteria were sufficiently prominent using factor analysis to construct a framework which provides a process that identifies within the requirements the criteria discovered. This framework is then evaluated for assessing suitability of a MAS architecture, by non-AOSE experts, on two real world problems: an electricity market simulation and a financial accounting system. Conclusion: Substituting a software engineer's personal inclination to (or not to) use a MAS, our framework provides an objective mechanism. It can supplant current practices where the decision to use a MAS architecture for a given problem remains an informal process. It was successfully illustrated on two real world problems to assess the suitability of a MAS implementation. This paper will potentially facilitate the take up of MAS technology.

Keywords

software, agent, framework, assessment, architectures, suitability

Disciplines

Engineering | Science and Technology Studies

Publication Details

Beydoun, G., Low, G. & Bogg, P. (2013). Suitability assessment framework of agent-based software architectures. Information and Software Technology, 55 673-689.

Suitability assessment framework of agent-based software architectures

Ghassan BEYDOUN, Graham LOW, Paul BOGG

Abstract—
Context: A common distributed intelligent system architecture is Multi Agent Systems (MAS). Creating systems with this architecture has been recently supported by Agent Oriented Software Engineering (AOSE) methodologies. But two questions remain: how do we determine the suitability of a MAS implementation for a particular problem? And can this be determined without AOSE expertise?

Objective: Given the relatively small number of software engineers that are AOSE experts, many problems that could be better solved with a MAS system are solved using more commonly known but not necessarily as suitable development approaches (e.g. object-oriented). The paper aims to empower software engineers, who are not necessarily AOSE experts, in deciding whether or not they should advocate the use of an MAS technology for a given project.

Method: The paper will construct a systematic framework to identify key criteria in a problem requirement definition to assess the suitability of a MAS solution. The criteria are first identified using an iterative process. The features are initially identified from MAS implementations, and then validated against related work. This is followed by a statistical analysis of 25 problems that characterize agent-oriented solutions previously developed to group features into key criteria.

Results: Key criteria were sufficiently prominent using factor analysis to construct a framework which provides a process that identifies within the requirements the criteria discovered. This framework is then evaluated for assessing suitability of a MAS architecture, by non-AOSE experts, on two real world problems: an electricity market simulation and a financial accounting system.

Conclusion: Substituting a software engineer's personal inclination to (or not to) use a MAS, our framework provides an objective mechanism. It can supplant current practices where the decision to use a MAS architecture for a given problem remains an informal process. It was successfully illustrated on two real world problems to assess the suitability of a MAS implementation. This paper will potentially facilitate the take up of MAS technology.

***Index Terms*—Modeling, distributed intelligent systems, agent, multi agent systems, software development**

I. INTRODUCTION

A distributed intelligent system is a collection of interacting intelligent individual components which cooperate to solve global goals as well as solving their local goals [41, 43]. The agency metaphor, as applied to such individual components, has proved fruitful in modeling their behavior and host systems. Indeed this has led to the acceptance of 'Agents' as highly

This work was supported in part by the Australian Research Council under Grant No.DP0878172.
Ghassan BEYDOUN is with The University of Wollongong, Wollongong, Australia (email: beydoun@uow.edu.au).
Graham LOW is with The University of New South Wales, Sydney, Australia (email: G.Low@unsw.edu.au).
Paul BOGG is with The University of New South Wales, Sydney, Australia (email: paulb@unsw.edu.au).

autonomous, situated and interactive software components. They autonomously sense their environment and respond accordingly. A distributed system formed from coordination and cooperation between agents is known as a Multi Agent System (MAS). The diverse knowledge and capabilities of individual agents within a MAS facilitate the achievement of global goals that cannot be otherwise achieved by a single agent working in isolation [73]. MASs have been shown to be highly appropriate for the engineering of open, distributed or heterogeneous systems [35, 38, 57].

Distributed MAS have been developed by Distributed Artificial Intelligence researchers since the 1980's. However it is more recent that many Agent Oriented Software Engineering (AOSE) methodologies have been proposed to guide the development of MAS (e.g. MaSE [27], GAIA [74], PROMETHEUS [52], MOBMAS [69] and TROPOS [13]). Such methodologies define various modelling languages, steps, techniques and models to produce MAS [3]. Whilst software architecture researchers aim at formalising the description of the system to facilitate the transition into design [48], software engineering researchers working on MAS architectures (aka AOSE researchers) aim to create requirement analysis concepts and tools to convert the problem description into agent based requirement models (to a varying degree of formalism). They often use different agent based constructs and target different development settings or phases. Gaia [76] for instance supports the development cycle of MAS from analysis to low level design. Prometheus [52, 66] defines an agent-based development process of three phases – system specification, architectural design and detailed design – to develop MAS based on a specific agent architecture (BDI architecture (Belief – the agent's knowledge of the world, Desire – the agent's goals, Intentions – the goals that the agent is committed to achieve at certain moment)). Adelfe [7, 8] is oriented to the development of adaptive MAS, i.e. systems that can adapt

themselves to unpredictable, evolutionary and open environments. PASSI [21] and its evolution ASPECS [22] focus on agent societies to describe a complete development process from requirements specification to implementation. TROPOS [2, 15] covers the analysis and design phases of MAS development and is based on the *i** requirements elicitation approach [75].

The notion of architecture in software engineering aims at reducing the cost of development [48]. Whilst MAS architectures can also contribute to cost management of a project [6], they are often pursued as a complexity management/problem solving tool which may in some cases allow tackling new problems [41, 43]. The decision to apply a MAS architecture and possibly an AOSE methodology to a given problem remains an informal ad-hoc process, based on the software engineer's inclination to use such architectures and/or past experience of using such architectures applied to similar problems. Given the small number of software engineers that are familiar with MAS and AOSE, many problems that could benefit from a MAS approach are solved with other approaches which may not be the best approach for a particular problem. This can indeed be in some cases an overlooked opportunity for a very productive and cost effective approach. Research has recently shown that, when suitable, MAS architectures can lead to large increases in the productivity of developers and programmers [6].

The lack of familiarity with MAS and AOSE has no doubt contributed to the delay in the much anticipated adoption of AOSE in many medium to large-scale projects. As pointed out in [1], as a particular technology matures, its accessibility to non-experts increases. We believe that AOSE has sufficiently developed and it is timely to facilitate access to this technology for non agent-experts. In this spirit and to overcome the above barrier to the successful adoption of AOSE, this paper provides a framework to guide a software engineer, who may not be an AOSE expert, to decide whether a MAS architecture may be an appropriate solution for a particular problem. The

selected development approach (e.g. *object-oriented*, *service-oriented*, *agent-oriented*) would depends on both the suitability of the resultant implementation as well as factors such as project cost and availability of experienced staff. We assume that the requirement gathering has been accomplished before the framework is used. In other words, we assume that any doubts about the requirements and the domain have been resolved. As such, to apply the framework, it is not necessary to have deep expertise about the application domain.

The rest of this paper is organized as follows. Section II describes related work and introduces previous attempts to establish types of problems suited for MASs. Section III proposes features of problems to which MASs are suited, based on an analysis of problems for which MASs have been developed. Further analysis on these features establishes relationships that indicate different problems have different degrees of suitability for MASs. Section IV uses this analysis to formulate a framework for determining the suitability of a MAS to a problem. This framework is placed within the context of a software development lifecycle. Section V describes an application of the framework to describe the suitability of a MAS to the problem of modeling an electricity market place for assessment of an electricity market simulation and a financial accounting system. Section VI concludes the paper.

II. Related Work

An increasing acceptance of AOSE as an alternative approach to software development has led a number of researchers to ask how and when AOSE would be preferred over other approaches [51, 72, 78]. This question has appeared prominently amongst researchers in agent-oriented development and is yet to be answered [78]. Towards finding an answer, the primary focus in this paper is in defining a measure of *suitability* defined as the extent to which a software solution adequately addresses the features of a particular problem domain. In reality, the answer

to the question *"Could one suitably use a MAS approach to solve this particular problem?"* could be based on an acceptable degree of suitability combined with an acceptable level of project and personnel-based estimates such as cost, time, and expertise.

Apart from a few application-based research works (e.g. cases presented in Table 5 later in this paper), none have demonstrated an analysis of suitability at the same level of depth of [37, 54, 55]. One of the earliest reviews of MAS suitability and general benefits can be found in [37]. In particular, this work analyzed the suitability of MAS for various telecommunication applications. The main conclusion was that MASs were suitable for telecommunication applications when key system requirements involve *distribution, robustness, responsiveness* and *flexibility*. This conclusion was later confirmed for other domains (such as logistics and space exploration) [54]. Similar conclusions were found in [55] for manufacturing and defense applications, but with additional system requirements of *openness, complexity* and *cooperative problem solving*. Whilst the analysis was convincingly thorough in those problem domains considered by [37, 54, 55], no domain-independent validations or generalizations were made. We believe that it has been briefly attempted only in AOSE in the context of creating new methodologies.

AOSE researchers directly involved with methodologies have attempted to formulate the suitability of MASs in trying to scope their methodologies. Notably [51] attempted this from the perspective of *"management, usage, and technical"* requirements. In their approach, they identified a small set of requirements, performed a survey-based validation, and compared two methodologies. In relation to MAS suitability, the technical requirements identified were: *legacy, distribution, environment, dynamic structure, interaction, scalability* and *agility.* This work is perhaps closest to the work presented here in that they identified suitability criteria and attempted a validation. However, their validation was limited to expert reviews and they did not

perform a comprehensive domain-independent analysis. Another notable related effort that appeared at the same time as [51], is the EURESCOM project [29]. This resulted in the following domain-independent characterization of when MAS is suitable:

- Complex/diverse types of communication are required;
- The system must perform well in situations where it is not practical/ possible to specify its behaviour on a case-by-case basis;
- Negotiation, cooperation, and competition between entities is involved;
- The system must act autonomously; and/or
- High modularity is required (for when the system is expected to change).

Similar suitability criteria to [27] were suggested by [25] with one addition:

- (when) There is decentralised or distributed information availability (e.g. in competitive situations, or communication failure somewhere).

Both [25, 29] provided criteria which are very broad and have not been validated. They do not constitute clear and comprehensive guidelines as to the suitability of a MAS-based solution. Since [29] appeared, some work indirectly assessing MAS suitability has been performed by comparing various AOSE methodologies e.g. [24, 59, 62, 64, 65, 68]. For instance, [24] identifies methodologies that may address problem domain-specific criteria for autonomy, concurrency and distributedness. Other examples include [65] identifying methodologies that address domain-specific criteria for communication and [64] identifying a methodology's applicability to multiple domains in terms of "expressiveness" which generalizes domain-specific criteria. However, it remains that [24, 59, 62, 64, 65, 68] have not provided criteria that assesses the suitability of a methodology (or MAS) for a given problem domain. These works provide particular views to assess methodologies e.g. in terms of agent concepts, modeling

notation, process-specific criteria and management-related pragmatics. The most important question to a software developer: "*Could one suitably use a MAS approach to solve a particular problem?"* remains unanswered and is our driving motivation in this paper.

In summary, research work towards assessing suitability criteria has been too general, incomprehensive, and largely unvalidated. Moreover, none has actually targeted producing knowledge that can be directly used by non-agent literate software developers. In this paper, we seek to correct what we see as a serious barrier to successful adoption of MAS software technology and associated methodologies. Rather than using our own experiences and judgments, or other experts' judgments, our starting point in this paper is to use a comprehensive set of existing MAS applications. By methodically analyzing this set, we synthesize a set of features detailing properties of problems suited for MAS. This leads to a more thorough analysis addressing further and more elaborate questions: to what extent is a MAS suited to address the features of a particular problem? Are some features more important in determining this suitability? And indeed, how can one validate the appropriateness of any defined set of features? In [12] we described a preliminary set of features that attempts to address these questions. In this paper, we extend the set identified in [12] and validate it against related work (Section II). This is followed by a statistical analysis of 25 problems that characterize agent-oriented solutions previously developed to group features into key criteria. We then construct a detailed framework that can be applied in a multi-staged fashion to coincide with the software development cycle of a specific problem instance to assess the extent of the suitability of a MAS solution.

III. MAS Features for Determining Suitability

It is well known that MAS properties result from the interactions between the individual agents in the system. Whilst it is relatively easy to ensure individual agents are developed with certain

desired features, it is more difficult for developers to ensure that MAS properties are consistent with their requirements. While agents are single components that can be well specified, MAS functionality is underpinned by the complex interactions of all the agents in it. In other words, it is not uncommon for the MAS to exhibit characteristics that were not foreseen and the development project could fail as a result. This section lays the foundations to construct a framework for assessing the suitability of a MAS to a given problem. It identifies features from problem specifications of implemented MASs. Later in Section IV, the results of the analysis are integrated into a new framework for evaluating the suitability of a MAS solution to any particular problem instance

An *individual agent* typically has varying levels of autonomy over its task execution and resource usage, social ability and reactivity enabling agents to interact with other agents or its environment, pro-activity and/or reasoning capabilities enabling it to use knowledge to guide its actions. As individual agents interact, collaborate and cooperate applying their reasoning capabilities, the system as a whole can change its internal organisation to adapt to changes in its goals and the environment without explicit external control [61]. This gives rise to various MAS characteristics (adapted from [61]): *decentralised control* governed by agent interactions*, self-maintenance* enabling it to repair itself*, adaptivity* enabling it to change as its environment changes*,* and *convergence* enabling it to eventually reach its goals and a stable state. For example, if the interactions between individual air conditioning unit agents keep the building temperature constant, then we might say the system has decentralised control, is adaptive to changes in the environment, and converges to a stable state. We seek to analyse a set of successful MAS applications to deduce features successfully implemented. We anticipate a strong overlap between them and the aforementioned characteristics that are typically ascribed to

MAS. However, systematically establishing a link between the features and the requirements is the essence of our goal in this paper. We adopt an iterative approach based on an initial analysis of 25 instances of problems for which MASs have been implemented. This approach consists of the following three steps:

A. Identify features that characterise problem instances where agent-oriented solutions have been implemented.

B. Validate and enhance the set of features identified in step A by comparing the derived set of features with those identified in existing frameworks described in Section II.

C. Develop and validate a features rating scheme

- Develop a ratings scheme for the features identified above.
- Validate and enhance of the features rating scheme developed in the first part of this step as follows:
    - Determine the adequacy of the feature set definitions and associated rating scale by employing an experienced software engineer to apply the feature set to a given problem.
    - Determine the repeatability of the rating scales by employing two independent assessors. Compare the independent assessors' ratings with existing ratings to identify differences and refine the rating scales, if necessary.

A. Problem & Solution-Related Features

We choose 25 applications which cover all technology areas identified by IEEE Spectrum (www.ieee.org). We ensure that we cover the following application areas: Aerospace, Biomedical, Computing, Consumer Electronics/Work, Energy, Green Technology, Robotics, Semiconductors/Manufacturing and Telecommunications. We identify 2-4 MAS applications in

each of the IEEE Spectrum areas (other than Biomedical where we only locate one such application). The chosen applications are all published works. In many cases, the users themselves are authors of the works. The users opinions are as such accessible and are used to evaluate the success of the application. It is deemed as successful only if a documented opinion of the users of the system is available and confirms its success. We deliberately do not start with the features included in frameworks identified in Section II – as this may inhibit us from identifying additional features and deny us the opportunity to use those for later validation. A set of features that characterise the spectrum of problems is identified. In deriving this set, every effort was made to make it *domain independent* and applicable to new domains not previously considered as candidates for agent-oriented solutions. We relate the resultant features to the properties of agents and 25 MAS applications and illustrate them using two MAS implementation examples:

An Intrusion Detection System (IDS) adapted from [4] *consists of mobile agents. These trace intruders, collect information related to the intrusion along the intrusion-route, and decide whether, in fact, an intrusion has occurred. Intrusions are mainly two types: break-ins from outside the local area network (LAN) and those from within the LAN. Intruders tend to attack the less-protected hosts first, gradually approaching hosts armed with stronger protection, ultimately working up to and reaching their target hosts. Commonly, administrators do not notice the intrusion. They also cannot trace the origin of an intrusion after the network connection has closed even when an intrusion has been detected. Attacks include data driven attacks on applications, host-based attacks such as privilege escalation, and malware.*

A Battlefield Information System (BIS) adapted from [46] *provides the commander with tactical and strategic intelligence during a conflict situation. To accomplish this goal, various*

types of sensors are used to detect events and objects of interest. This sensor data can then be combined, or fused, with other sensor data to provide a commander with a much clearer, more complete picture of the battlefield. Due to the nature of war, there is also a high probability that a percentage of these sensors will become disabled during the battle. However, when those are lost or destroyed, the information produced by those sensors must still be provided to the battlefield commander.

Software solutions operate within an *environment*. This is a computer system, network or existing software within which the software solution is embedded. During the execution of a process, the software solution may require information or resources from their environment. Properties of the information or resources are themselves features related to the problem. We identify the following:

1. *Uncertain environment* – It may or may not be *known* whether information is accurately acquired by the software. When it is not known that it is "100% accurate", information is said to be "uncertain". Information may be uncertain when it is "old", may have changed, is provided by an unreliable source, or when it is acquired by some means of approximation (as is done with computer visual processing). This notion of 'uncertain environment' is similar to the *observability* notion in [58]. Here however, our focus is on the environment itself rather than on the agent perspective which would lead to uncertainty in the environment to be evident as a limit on what agents can or cannot observe. In either case, agents required to perform tasks in an uncertain environment can *reason* about the uncertainty of the information and resources required. As *autonomous* entities, agents have control over the management of uncertain information or resources in order to achieve their goals. As *social* entities, agents may confer to reduce uncertainty.

2. *Dynamic environment* – During the runtime of a process, information or resources received by the software may change. This may be frequent, e.g. information from the stock market, or this may be infrequent, e.g. prices of digital cameras. Agents required to perform tasks in dynamic environments need to be *reactive* to changes and often need to *reason* about those changes. As noted in [58], the result of an agent's reasoning can quickly become out of date in a dynamic environment. As *autonomous* entities, agents have control over what to do as a result of undesirable or unexpected changes in order to achieve their goals. MASs have *adaptive* behaviour that adjusts to environment changes. For example, the battlefield simulation system adapts to the disabling of sensors in order to continue operation.

3. *Dependable environment* – During the runtime of a process, the software may depend on the uptime of some components that provide information or resources. For instance, the software may depend on the uptime of humans, servers, databases, and so on. Agents that perform tasks in undependable environments may need to *reason* about alternative courses of action when components are down, and as *autonomous* entities, have control over what alternative action to take. MASs need to *adapt* to the downtime of components in an environment.

4. *Open environment* – During the runtime of a process, new software components might be added to the existing software. The software might be required to facilitate or adapt to the new software components. Agents required to perform tasks in an open environment can *reason* about new software components and the changes to the information and resources that they bring. As *social* entities, agents may communicate with new software components or with other agents about new components. MASs may need to be *adaptive* to new software components in order to preserve system operation. For example, an IDS adapts itself to monitor new computers on the network to maintain system-level security.

5. *Distributed environment* – The information or resources that are acquired by the software during runtime might be distributed across a network. For instance, information/resources might be highly distributed, scattered across many servers across the world. Agents may acquire geographically distributed information/resources in a *proactive* or *reactive* way. As *social* entities, agents may exchange information/resources with other geographically distributed agents. MASs can have *decentralised control* over the acquisition of geographically distributed data. For example, the battlefield simulation system uses sensors distributed over a wide geographic area to exchange data when one or more sensors are disabled.

6. *Communication quality* - The communication of information or resources to the software might not be reliable. In networks (cable or wireless), the communication medium may determine the quality of the communication of information/resources in terms of dropout rates and noise.

In both examples, the IDS and the BIS, the input information from the environment is *dynamic*. The extent of this dynamicity depends on the environment for which the system is developed. For networked systems, an IDS may be required to monitor attacks on new computers or network components added at runtime. A difficulty in designing an IDS is that any solution is required to handle repeated and new forms of attack, as such the input from the environment may also be *open*. Similarly for the BIS, existing sensors may be likely destroyed and/or replaced and the environment needs to be also open. Furthermore, a feature of the BIS is to handle new events and objects of interest. The BIS is situated in a *distributed environment*. While the sensors used to acquire the data might be distributed, a significant proportion of tasks would be performed centrally.

We also identify a set of requirements that are themselves features of the system.

7. *Efficiency* – this is any consideration given to the degree to which expenditure of time and resources is minimized in order to perform the tasks required. As autonomous entities, agents may provide efficiency by having complete control of their inputs and resources. For example, an autonomous Mars rover reduces the communication control overhead from Earth. Agents may also reason about how to perform tasks more efficiently. MASs that have decentralised control and adaptivity provide efficiency by reducing the overhead that would normally be required of a centralised control to adapt and regulate variables in response to changes in the environment. For example, an IDS is efficient in the sense that new attacks are detected by local agents in order to maintain global security.

8. *Robustness* – This is any consideration given to the continuity of performance of certain tasks under adverse conditions e.g. computational failure or communication failure. As autonomous entities, agents may provide robustness by having control over self-maintenance tasks in adverse events (the Mars rover might be capable of self-healing [25]). MASs that have decentralised control, adaptivity and self-maintenance provide task robustness by adapting to parts (or agents) of the system that malfunction e.g. a battlefield simulation system with decentralised control is robust in operation because it adapts to the loss of sensors.

9. *Reliability* – This is any consideration given to the continuity of task performance as expected, without error/mistakes. As autonomous, reasoning entities, agents may provide reliability by reasoning about environment changes that may affect task performance. MASs that have adaptive behaviour provide reliability in regulating system-level behaviours when the environment changes.

10. *Flexibility* – This is any consideration given to the ability of the software to perform new tasks, and cope with any changes to existing tasks at run time. As autonomous, reasoning

entities, agents may provide flexibility by reasoning about new and different tasks to be performed, and having control over when to perform them. MASs may have adaptivity that adjusts the system operation to new problems or tasks. For example, the MAS for an IDS provides flexibility in detecting new forms of attack

11. *Responsiveness* – This is any consideration given to the speed at which tasks are performed. In particular, when tasks are prompted to be performed by external/internal components. An agent is reactive to environment changes that affect task performance. As an autonomous entity, an agent has independent control over how it reacts, which improves its responsiveness compared with a centralised controlled approach. MASs may have decentralised control that improves the responsiveness of the system due to less overhead in requiring checks with a centralised control.

12. *Indeterminate* – This is any consideration given to when it is not known what tasks to perform, or how to perform them. In other words, how much consideration is required towards designing software that can make its own decisions as to what to do at runtime. As autonomous, reasoning entities, agents may reason about what task is best to perform, and have control over the performance of this task.

13. *Concurrency* – This is any consideration given to tasks that are to be performed simultaneously. MASs as a composition of autonomous, reasoning agents may perform tasks that are required to be concurrent.

14. *Scalability* – This is any consideration given to the ease with which the performance of tasks can be extended to new systems/environments in the future. MASs, as adaptive, decentralized systems, are capable of being extended for operation in new environments.

15. *Distribution (of tasks)* – This is where tasks need to be performed on different computer(s) in

geographically separate locations. (In the case where information/resources are distributed, but the tasks to acquire the information/resources are not distributed, the tasks are not considered as a distributed feature). Agents may be proactive or reactive in performing tasks in remote locations. As social entities, agents may ask other agents to perform tasks remotely. MASs have decentralised control over the execution of tasks, meaning other agents in remote locations may be asked to perform tasks. For example, the IDS uses mobile agents to perform remote security-based tasks in a distributed network.

16. *Legacy* – consideration given to integrating tasks performed by old existing software, where this software is not redesigned, with the new software. Legacy software might warrant consideration given to "wrapper" software as an interface to new software. Agents may be used as this "wrapper" software.

17. *Interaction dependencies* – The performance of a task may depend on the interactions with external components (humans or other software). The performance of various tasks may be interdependent. Consideration may be needed towards designing software that is capable of this interaction.

18. *Interaction type* – Software solutions often need to interact with other software, or components within their environment. Interactions might be as simple as an enquiry for information, or as complex as a negotiation. The properties of these interactions are themselves features required of the solution (as adapted from [70]):

a) Simple enquiries – requesting/receiving information e.g. yes/no type questions.

b) Negotiation – the process of coming to agreement on a set of issues (for example, establishing communication protocols, or placing bids on auctions). Agents may be proactive or reactive in the process of negotiation in order to achieve their goals. As a

social entity an agent may negotiate with one or more agents. As an autonomous entity, an agent may have control over what information is relevant to the negotiation. When negotiation is needed to regulate the amount of resources amongst entities, MASs may have convergent behaviour that regulates how resources are distributed e.g. the stabilising of a global market price amongst competing agents in a market place [23].

c) Cooperation – where software components need to cooperate with one another in order to perform a task/process, requiring negotiation. As a social entity an agent may cooperate with one or more agents. As an autonomous entity, an agent may have control over what information is relevant to the cooperation. For example, agents may cooperate in dynamic load sharing for CPU intensive tasks.

d) Argumentation – where software components engage in debate, dialogue and negotiation to establish agreements and/or acceptable conclusions. For example, agents may be used to resolve conflicts in law. As a social entity an agent may engage in debates with one or more agents. As an autonomous entity, an agent may have control over what information is relevant to the debate.

e) Complex interactions – some combination of negotiation, cooperation, and argumentation. For example, software automating an economic business process may require negotiation and argumentation with external business partners to acquire goods and services. In the BIS example, when sensors are destroyed, some negotiation and cooperation between the remaining sensors is required in order to compensate.

Although we define features independently from one another, often in real problems they are conjoined. For example, an IDS requires that the system be reliable in continuing to detect known and new forms of attack. Robustness is required in situations where network

components fail. Flexibility is required in detecting new forms of attack. Responsiveness is imperative in order to promptly alert the system administrators of possible intrusions. Concurrency is required when many parts of the network are monitored simultaneously. There is an expected compromise in general efficiency; however, new forms of attack are required to be detected efficiently. Moreover, tasks are required to be performed remotely and there is a need for distributed tasks. (The feature of task distribution might be dependent on the framing of the problem. A similar IDS problem might not require distributed tasks.)

Similarly for the BIS example, the system should be reliable so that it can continue to monitor the battlefield. Robustness is imperative since sensors can fail. Responsiveness is imperative for providing information. However, flexibility, handling indeterminism, and high efficiency are not necessarily solution-related features for this problem instance.

B. Validation against existing Frameworks

The feature identification in Step A may be biased by the chosen examples and the way each problem was originally framed. Hence, we add a cross validation step in this section. The resultant features are cross checked to ensure they cover criteria described by [25, 29, 37, 51, 54, 55, 73] from the related work section (Section II). Our intention throughout the work is to ensure that the framework is usable by software developers who do not necessarily know a great deal about agent technology. Results of this cross validation are shown in Table 1. Our features address most criteria described by [25, 29, 37, 51, 54, 55, 73], with the exception of a few excluded for either of two reasons: First, some are excluded as they are subsumed by one or a combination of our suggested features. For example, our feature set does not include '*complexity*' which in [37] refers to problems being too large for centralized solutions. Using our features set this can be said to require a high degree of *distribution*. Second, other features were intentionally excluded as they typically describe actual agent-oriented systems, rather than

general requirements of any system. Their use would require specialized agent knowledge and would be too cumbersome for a non-agent expert software developer. These are too difficult to clearly conceptualise in a general requirements engineering phase. One such example feature is *'if agents are a natural metaphor'* from [73].

Some other features were excluded due to a combination of both reasons. They were too agent-specific and at the same time could be subsumed by one or a combination of our existing features. For example, '*autonomy*' from [54] is an agent-oriented system feature that is typically desired by agent designers in situations where communication with a software component is not continuously guaranteed thus making central control less desirable in a distributed environment. System requirements could in this case be better articulated in terms of our general features of *concurrency*, *distributed(ness)* and *indeterminate(ness)*. Another similar example is the exclusion of '*self-maintenance'* from [25] which is a MAS desired feature which can be better articulated by a software engineer using general features of *robustness* and *reliability.* In these instances, our general features are preferred to the features of agent-oriented systems to maintain consistency with supporting software developers unfamiliar with agent technology.

**** Table 1 about here ****

C. Develop and Validate a Features Rating Scheme

In the first part of this step, we develop a rating scheme consisting of specific definitions for each feature and an associated rating scale for a software engineer to elicit the presence (or degree) of a feature in a particular problem, i.e. its *pervasiveness*. As an example, to determine the *pervasiveness* of robustness (*consideration given to the continuity of performance of certain tasks under adverse conditions*), we ask: What consideration is needed towards how robust the software should be?

1. None – the software does not need to be robust

2. Small – a few tasks should be robust in performance
3. Moderate – a number of tasks should be robust in performance
4. Widespread – most tasks should be robust in performance
5. Whole – the whole system should be robust in performance

The validation includes both the validation of the enhanced feature set and its associated definitions as well as the assessing the repeatability of the ratings scheme.

Validation of feature set

To further ascertain the adequacy of the feature definitions and the associated ratings scheme, we asked an experienced software engineer to apply the proposed feature set on a Call Management system for a service support centre. The principle aim of the proposed system was to route incoming calls to appropriate human operators in accordance with a customer profile, generate statistics on overall calls taken, and to some extent, try to solve the customer's problem before it reaches a human operator. Application of the feature set was performed during the requirements stage. In total, there were four iterations over a period of approximately 6 weeks with refinement of the definitions and associated ratings scales at each stage of the process. More information on the ratings for the call management system can be found in [5].

Repeatability of the rating scheme

Repeatability of the ratings scheme was determined by comparing the ratings assigned by two independent experienced software engineers and one of the authors to six problem instances in the literature (Table 3). Table 2 shows the average difference and variance in ratings for each feature across the six problem instances analysed by the two independent assessors. Table 3 shows the average difference and variance across all feature rating sets for each of the six

problem instances[1].

**** Table 2 about here ****

At the *individual feature* level, there were three features that had an average rating difference >0.35: Uncertainty, Dependability, and Flexibility. Discussion with the independent assessors indicated that it was difficult to accurately determine the impact of these features for a problem domain. There was a consensus on the presence of each feature, however, opinions on the pervasiveness of the feature varied due to a lack of precision in their definitions. The definitions of these features were subsequently revised to overcome this problem.

Three features had a variance in rating assignment >0.05: Openness, Flexibility and Indeterminism. It was determined previously that there was a difference in opinion on the pervasiveness of Flexibility that contributed to the variation in ratings between the two independent assessors. For both Openness and Indeterminism the variance may be attributed to differences in understanding the problem instances. For example, in the case of the Fish Auction instance, one assessor considered openness a feature of the problem while the other considered it a closed environment. For Indeterminism, the discrepancy in the Document Recommendation instance was 0.6 (whereas, the average was 0.267) – each software engineer had conflicting interpretations of the problem instance for whether indeterminist behavior was required.

Features not explicitly described in the requirements showed more variation in usage. In particular, uncertainty, dependability and openness feature ratings could only be implicitly identified – hence, there was greater scope for variation in ratings elicitation. However, there was good consistency between the six problem instances. Minor rating discrepancies could be

[1] In order to summarise the data, we provide a simple distance function that measures the average distance between the ratings across all of the application domains. For an application domain (e), for problem feature (f), rating range R={1,2,3,4,5}, the author's ratings ($Ra(e,f) \in R$), the participant's ratings ($Rp(e,f) \in R$), and the max rating possible (m) are used to define the distance between ratings as: $dist(e, f) = |Ra(e, f) - Rp(e, f)| \ / \ m$

attributed to a difference in opinion on pervasiveness in the feature definitions previously described. The problem instance for which the rating scheme was most consistently applied was Manufacturing. One explanation for this is that the Manufacturing instance was the most straight forward problem instance to which a software-based solution could be found – whereas other problem instances had greater scope for variation in the possible types of solutions.

**** Table 3 about here ****

We also compared the independent assessors' ratings against those of one of the authors. Overall, there was good consistency across the six problem instances, described in Table 4. Features found to have a rating difference >0.35 were: Dependability, Openness, Legacy, and Distributed Tasks between the author and assessor A; and Uncertainty, Efficiency, Flexibility, Legacy, and Distributed Tasks between the author and assessor B. As noted earlier in this section, the probable cause of variation in the ratings for Uncertainty, Dependability, and Flexibility was the difficulty in determining the pervasiveness of the feature. For Openness, it had been determined that the cause of variation was due to differences in interpretations for the Fish Auction problem instance. Further retrospective analysis was conducted to determine the reason for the variation between the independent assessors' and the author's rating for Efficiency, Legacy and Distributed Tasks. With respect to Efficiency, there appeared to be a minor difficulty determining the rating from the problem instance descriptions provided. For Legacy, the need for legacy systems was not commonly described in problem instance descriptions. In researching for the problem instance description the author had a different, possibly broader, understanding of a problem than the independent assessors. This was found to be have been a factor in the differences experienced for the features, Legacy and Distributed

Tasks – the latter particularly in the Document Recommendation description.

This analysis highlights an important aspect of the rating scheme application: when an independent assessor is provided with the feature definitions and a problem instance description, he/she is likely to identify similar ratings as another independent assessor – presuming the problem instance description is consistent.

**** Table 4. about here ****

In the following section, we use the analysis of this section to define a framework for suitability that can be interleaved within a software development process.

IV. Framework for Determining MAS Suitability

In Section III, we outlined a set of features that characterise a general set of problems. We also related specific properties of agent-oriented solutions to specific features of the problems. In this section, we develop a framework that assesses the suitability of a MAS solution given a specific problem. This framework takes a problem instance and provides guidance as to the suitability of an MAS solution. The development process involved two steps.

A. Identify feature groupings and suitability criteria.

B. Develop the framework using the suitability criteria from step A.

A. Feature groupings and agent oriented suitability criteria

In Section III, we outlined a set of features that characterise a general set of problem instances suitable for an agent-oriented solution. One or more of the features of the problem may correlate with the desirability of one or more properties of the resultant agent oriented system. Some of the properties of agent-oriented system itself, as earlier discussed, are due to properties of the agents themselves, and some are due to their collective interactions as a MAS – referred to as *agent-level* and *system-level* properties respectively [9-11]. Some features are more prominent than

others, depending on the problem. Table 5 illustrates the ratings given (1-5) for each feature on the extent to which it appropriately described each of the 25 problem instances analysed from the literature. Before we outline the framework for determining MAS suitability in Step B, we undertake a deeper analysis of the set of features identified investigating any underlying common dimensions (or factors) in the set of features. We apply a statistical tool, *Factor Analysis,* to find a way of condensing the information contained in a number of original variables (features) into a smaller set of dimensions (factors) with a minimum loss of information [36]. The results (Table 6) indicate 5 underlying factors. Intuitively the factor groupings make sense. For instance agent-oriented systems were developed for problems that generally required components to operate concurrently and where the system depended on the interactions between those components. In many cases, the type of interaction required was complex – i.e. cooperation or negotiation. Agent-oriented systems were also developed for problems that generally required solutions to operate in dynamic and uncertain environments. In some of these cases the environment or system components were also (to a degree) undependable. Not surprisingly "legacy systems" is assigned to its own factor. While a MAS based solution may assist in interfacing with legacy systems, its presence would not necessarily imply the existence (to any large extent) of any of the other features identified.

The mean and standard deviation of the problem and solution-related features for the 25 problem instances analysed is shown in Table 7. Those features ranked highest were uncertainty, interaction dependencies, responsiveness, reliability and concurrency. Interestingly the mean rankings of the various features for the first 3 factors are generally much higher than those for factors 4 and 5.

**** Table 5. about here ****

**** Table 6. about here ****

**** Table 7. about here ****

Factor analysis provides an insight into the underlying common dimensions of features. We look for feature groupings within the individual factors that may have specific meaning. An examination of the factor analysis and associated Pearson's correlation matrix suggests the feature groupings shown in Table 8. The groupings are based on "large" positive Pearson correlation coefficients between the features (0.5-1.0). Each feature grouping is associated with only one factor as indicated in the table.

**** Table 8. about here ****

These groupings suggest that, for problem instances where MASs have been successfully implemented, particular features tend to be associated. Based on these feature groupings we suggest five suitability criteria to determine if an agent-oriented solution is suitable to a given problem instance.

1. *There is a necessity for the continual successful operation of software situated in dynamic, uncertain and undependable computing environments*. – based on feature grouping 1. Undependable environments can lead to dynamic environments as information or resources availability frequently change, and this in turn leads to uncertainty as

information or resources availability is no longer known. Where only one of these three features is present, a non-agent-oriented solution may be more appropriate. Where two or more of these features are present, an agent-oriented approach may be more suitable. For instance, expert systems may be employed to handle decision-making in uncertain environments. However, expert systems may not successfully cope in dynamic and uncertain environments.

2. *There is a necessity for independent, concurrent software components to automate a process where interoperations between new and existing components are required.* – based on feature grouping 2.

   Open environments require (possibly new) independent software components to interoperate with existing software components during runtime. When these independent components operate concurrently and the interoperation is sufficiently complex, a MAS solution may be appropriate. For instance, designing a Fish Auction market simulation requires independent software components to operate together. One possible solution is enforcing global turn-by-turn operations between components. But with concurrency and successful runtime interoperation with new software components, it may be more suitable to have independent software components coordinate auction operations on their own.

3. *There is a necessity to ensure the continual, successful operation of distributed software* – based on feature grouping 3.

   When software is required to be distributed, it may be desirable to ensure the continual, successful operation of the software over time ie. it may be required to be robust, reliable, and/or scalable. Designing distributed software may not necessarily mandate

assurance of its continued successful operation. However, where fault-tolerance is a necessity in distributed software, then MASs may be more suitable to ensuring continual, successful operation. For instance, it may not be critical that peer-to-peer file sharing clients operate without failure. However, in designing software for a Mars rover, robustness and reliability are essential requirements for performing distributed tasks.

4. *There is a necessity to ensure successful operation of software where communication quality is expected to be low* – based on feature grouping 4.

   For problem instances where communication quality is expected to be low, it may be desirable to ensure successful operation of a software solution by explicitly focusing on a requirement for responsiveness.

5. *The software solution is required to determine how to perform tasks, which tasks to perform, or integrate new tasks during its runtime* – based on feature grouping 5.

   Software required to handle indeterminate tasks may also be required to handle new tasks introduced at runtime. Handling new tasks is, in a sense, also an indeterminate task. For instance, expert systems may be employed to determine tasks performance during runtime, assuming that prior knowledge exists about how decisions are made. However, introducing new tasks or introducing new problems for which decisions are to be made requires software capable of reasoning (or perhaps learning).

Feature groupings 6 and 7 do not suggest any additional criteria. They are features (Efficiency and Legacy respectively) that are, generally speaking, no better addressed by an agent-oriented system than a non-agent-oriented system. However, a legacy wrapper might be required to address additional features (such as Criteria 2), in which case an agent-oriented solution might be more suitable.

B. Assessment framework

We now present a three step framework which takes a problem instance and provides guidance as to the suitability of an AOSE solution.

1. Rate the pervasiveness of each of the features (listed in Table 9).
2. Classify each feature into one of the associated 5 criteria for suitability (see Section IV A and Table 9).
3. If all the features of a criterion (or associated feature grouping) are rated as moderate or above, it is recommended that an agent-oriented analysis be considered. If only some features of a particular criterion are rated moderate or above, it may indicate that an agent-oriented system is suitable, but alternative approaches might just as likely be suitable as well.

Table 9 provides an easy way for the software engineer to present the results of steps 1 and 2 and to assist the software engineer in assessing the suitability of an AOSE approach to a particular problem instance (step 3).

**** Table 9. about here ****

MAS architectures provide a problem solving capacity suited for complex problems that are distributed and have a certain level of uncertainty and indeterminism. The framework assists the decision whether to implement a solution to a particular problem using a MAS. Should a MAS solution be chosen then the software engineer may use an AOSE methodology (e.g. MaSE [27], GAIA [74], PROMETHEUS [52], MOBMAS [69] and TROPOS [13]). Implementing a MAS architecture can benefit from a specific requirement analysis approaches provided by such methodologies to transform the requirements into appropriate models that can then be used to

derive a MAS. An architectural design approach [32, 48] or a pattern oriented software architecture approach (e.g.[14, 28]) are also possible. The approach selected is beyond the scope of the current research. With this said however, in our opinion, the notion of architecture that is commonly used in software engineering, needs further qualification for MAS: Commonly, architectures facilitate the design of the system and the transition from requirement models to design models [48]. In MAS development, the early choice of the architecture should change the requirement analysis process and how the requirement models are first synthesized.

We envisage that our framework can be applied in different phases of the development process. During the *requirements* phase, problem features may become known to the developers and the framework may assist in determining if a MAS solution is a suitable option. If so software developers may select an appropriate analysis methodology (e.g. according to [68]). During the *analysis* stage, the significance of various features may change, and a reapplication of the framework may influence the decision to continue (or discontinue) on with an agent-oriented design. If the framework suggests that an agent-oriented system is suitable, software developers may continue with an agent-oriented design methodology (e.g. according to [17]). By the start of the *design* stage, developers are reasonably certain of the suitability of their chosen approach. Nevertheless, a reapplication of the framework might still be necessary if important requirements change or new ones are added.

V. Application of Framework

In this section we illustrate our proposed framework to determine the suitability of an agent-oriented software engineering solution to the Australian National Electricity Market application (NEM) as well as to a financial accounting application. Ratings were performed by one of the authors and an independent software engineer for both systems.

NEM Simulation

The National Electricity Market supplies electricity to approximately 8.7 million residential and business customers. It is a wholesale market through which *generators* and *retailers* trade electricity across state borders. The six participating states are linked by an interconnected transmission network. The NEM is a wholesale pool into which generators sell their electricity. The main customers are retailers, who buy electricity for resale to business and household customers. It is possible for end-use customers to buy directly from the pool, but few choose this option. Generators earn their income from market transactions. Whilst electricity demand varies, industrial, commercial and household users each have relatively predictable patterns, including seasonal demand peaks related to extreme temperatures. NEM uses data such as historical load (demand) patterns and weather forecasts to develop demand projections. Demand peaks occur in summer (airconditioning) and winter (heating). Generator offers are affected by a range of factors: plant technology (coal, gas, renewable, etc), ramp rates (how quickly generators can adjust their level of output), and congestion in the transmission network. The simulation model is required to factor in regional and temporal demand forecasts, plant generator performance, new forms of electricity generators, existing interconnection limits, and the potential for new interconnection development.

The overview structure of the National Electricity Market Simulation [33] is shown in Figure 1.

*** Figure 1 here***

We will now use our framework to assess the suitability of a MAS-based approach for this problem. An agent-oriented system may be able to create simulations of an electricity market whereby market participants (generators, retailers, and end-user customers) buy and sell

electricity via auctions. We apply our framework to assess whether an agent-oriented development approach is suitable (ratings shown in Table 10).

**** Table 10. about here ****

To complete the assessment of whether an agent-oriented development approach is suitable, we now assess the electricity market domain against each of the five criteria:

Criterion 1. The electricity market domain has a highly dynamic environment. Information regarding electricity demand, generator output, connectivity changes, and market players is in a constant state of flux. However, there is only a moderate degree of uncertainty and information about the electricity infrastructure is reasonably dependable. Demand variations are predictable and often in accordance to weather seasonal changes. The actual electricity output from generators is reasonably stable. Dynamism coupled with moderate uncertainty in the environment indicates that an agent-oriented approach may be suitable, but the rating of this criterion alone does not deliver a conclusive indication.

Criterion 2. The environment of the domain is moderately open. It is presumed that participants may enter and leave the electricity market, and new retailers and generators may evolve. But it is not completely open in the sense that there only three (general) types participants are present – generators, retailers, and end-user customers. Generators, retailers, and end-user customers must interact for electricity trade between each other. This trade involves automated negotiation (principally via auctions) and some degree of cooperation. Concurrency is a necessity for auction bidding and selling, due to real-time needs of end-user customers. This need for concurrency together with a strong requirement for interaction dependencies between

the market participants suggests that an agent-oriented solution is (very) suitable for this problem. Agent-oriented solutions would also support the need to simulate a moderately open environment.

Criterion 3. The electricity market is distributed, however, most market trade occurs centrally. The computational model would need to account for a moderate degree of distribution in simulating the behavior of the participants. The electricity generation is required to be robust, as is the electricity market itself. The simulation addresses the need for robustness. The electricity generation and market are also required to be reliable. The electricity market in Australia is expected to grow but the scalability need is limited because of the relatively small number of players. The presence of the requirement for robustness and reliability suggests that an agent-oriented system might be suitable. The moderate degree of distribution and scalability however suggests that this criterion (on its own) does not conclusively suggest the suitability of an agent-oriented solution based on this criterion.

Criterion 4. The communication quality and responsiveness are not pervasive requirements for simulating the electricity market. Hence, this criterion does not suggest an agent-oriented solution.

Criterion 5. There is a moderate need to automate tasks that are not completely known at design time: There is a reasonable amount of data to estimate seasonal demand, generator plant performance, and infrastructure changes in order to simulate how market participants may behave. For this reason, participant behavior is to a degree determinate at design time. However, due to the complexity of factors used in decision-making by market participants, it is not completely known how they would behave, and so market behavior is (by nature) indeterminate. There is also a moderate need for the system to also address the flexibility of an

electricity market environment in adjusting to new types of participants and new factors that affect market behavior. These two moderately pervasive features suggest that an agent-oriented solution might be suitable. However, the rating of this criterion alone does not deliver a conclusive response to adequately suggest agent-oriented solution suitability.

In summary, the strength in presence of rating responses for each of the 5 criteria presents an argument as to the suitability of an agent-oriented solution to simulating an electricity market and participants' trading behavior within this market. The most prominent criterion for suggesting agent-oriented solution suitability was criterion 2: *the necessity for independent, concurrent software components to automate a process where inter-operations between new and existing components are required.* Less prominent criteria were criteria 1 and 3: *the necessity for the continual successful operation of software situated in dynamic, uncertain and undependable computing environments* and *a necessity to ensure the continual, successful operation of distributed software*. Although features of the latter two criteria were less prominent, they remain critical to be addressed by a software solution. Applying the framework identifies a number of criteria sufficiently prominent to suggest that an agent-oriented analysis is suitable. The elicitation of the ratings for the domain criteria is still preliminary, but it can assist software engineers in formulating an idea of what computational approach might be best suited to address the problem. A repeated application of the framework post-analysis stage, when problem-related and solution-related features are better known, may yield a clearer indication of agent-oriented system suitability.

The finding that an agent-oriented solution is suitable is consistent with the NEM Simulation being implemented with an agent-orient approach [33, 34]. In addition there are other examples in the literature demonstrating the viability of both agent-oriented and non-agent-oriented

approaches to simulating electricity markets e.g. [18] uses a non-agent computational approach whilst [71] and [41] provide an overview of agent-based approaches. The decision of a best suited computational approach is actually context dependent and may also further evolve during the software development process. Having a framework to assist in such a decision would, in the authors' opinion, lead software developers to deeper and more thoughtful reflections in developing their system analysis models, architecture and designs. This would broaden their modeling options during the software development process and most likely lead to better quality software (agent or non-agent based).

Financial Accounting System

The financial Accounting application [20] aims to collect, process and report information related to financial transactions. It provides information on the organisation's financial condition including the composition of its assets and liabilities, periodic operational results, and transactions with customers and creditors.

We apply our framework to assess the suitability of an agent-oriented approach to this problem (Table 11). Neither expert rated any of the features as moderate or above; in fact most features were rated as very low (i.e. value of 1). Since the assessment framework only recommends consideration of an agent-oriented solution if some or all of features of a criteria are rated moderate or above, an agent-oriented solution is not recommended for this system. This suggests that a non-agent oriented approach may be a more suitable approach. Given the central nature of financial systems, the outcome of the framework is not surprising. This is reassuring since accounting-type applications are usually implemented using an object-oriented approach.

VI. Conclusion and Future Work

This paper firstly abstracts key features from a variety of different problem domains. Features

are iteratively pruned to identify those significant determinants of whether or not a MAS is suitable. It secondly constructs a framework which assesses the suitability of a MAS solution, given a set of problem related features. Whilst our feature identification may be biased by the chosen examples and the way each problem was originally framed, resultant features actually generalise (and address) all criteria described by [25, 29, 37, 51, 54, 55, 73]. Since it is the intention that the framework can be applied by software engineers who may not be experts in AOSE, the features included in our framework are determined from the problem itself and *not* the properties of the resulting solution. The features were derived from a set of various MAS applications. For example, the framework has features that lead to identify complex interactions e.g. as in social organizations in defense applications. However the software engineer applying the framework does not need to have an understanding of the possible MAS implementation options.

We validated both the completeness of the proposed feature set and its associated ratings scale in Section III C. The average difference and variance in ratings between two experienced software engineers suggests that the rating scheme was applied consistently by each assessor, even if the final rating was slightly different. Provided the same person applies the rating scheme, there appears to be internal consistency in the resulting rating set.

Our framework can be used where alternative ways of addressing a solution are considered. For example in the Mars rover scenario, self-healing might be considered when *robustness* is necessary in performing *distributed tasks* in the event there is a problem. However, self-healing is not the only means of satisfying the need for robustness – an alternative means might be to have redundancy (many of the same system operating). Steps 2-4 in the proposed framework allow the designer to consider the appropriateness of a MAS solution for both these means of

satisfying the need for robustness. In addition our framework, being domain independent, may be applied to scenarios in which MASs have yet to be applied in order to determine its suitability.

Our analytic framework is subject to on-going validation. We intend applying our proposed framework to other problem instances to refine the framework as well as the feature descriptions and their associated ratings and to ensure that we have not missed any key problem and see if we can suggest weightings for the various features As part of our validation process and to demonstrate/improve the appropriateness of our proposed framework we plan to apply it to other problem instances where a MAS based solution has been implemented as well as problem instances which have not yet resulted in MAS applications. Further, structured interviews with experts in AOSE will also confirm the appropriateness of the identified features.

We also intend to further examine the consistency of the ratings scheme between different raters by recruiting a wider selection of software engineers to perform the ratings. The current study is limited to two raters although every effort was made to minimise internal validity issues. For instance both raters were provided with identical documentation on the problem instances and instructions on the application of the ratings scales.

REFERENCES

[1] R. Ahmad, S. Rahimi, Motivation for a new formal framework for agent-oriented software engineering, International Journal of Agent-Oriented Software Engineering 3(2) (2009) 252-276.

[2] R. Ali, F. Dalpiaz, P. Giorgini, Location-Based Software Modeling and Analysis: Tropos-Based Approach, in: Q Li, S Spaccapietra, E Yu, A Olivé (Eds) Conceptual Modeling - ER 2008. Springer Berlin / Heidelberg, 2008, pp 169-182.

[3] E. Argente, G. Beydoun, R. Fuentes-Fernández, B. Henderson-Sellers, G. Low, Modelling with Agents, in: M-P Gleizes, J Gomez-Sanz (Eds) Agent-Oriented Software Engineering X. Springer Berlin / Heidelberg, 2011, pp 157-168.

[4] M. Asaka, S. Okazawa, A. Taguki, S. Goto, A Method of Tracing Intruders by Use of Mobile Agents, 9th Annual Conference of the Internet Society, 1999, pp. 98-108.

[5] A. Ashamalla, Beydoun, G. and Low, G.C., Agent Oriented Approach to a Call Management System, 18th International Conference on Information Systems Development (ISD 2009), Nanchang, China, Springer, 2009, pp. 345-356.

[6] S. S. Benfield, J. Hendrickson, D. Galanti, Making a strong business case for multiagent technology, Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, Hakodate, Japan, 2006.

[7] C. Bernon, M.-P. Gleizes, F. Migeon, G. Marzo Serugendo, Engineering Self-organising Systems, in: G Di Marzo Serugendo, M-P Gleizes, A Karageorgos (Eds) Self-organising Software. Springer Berlin Heidelberg, 2011, pp 283-312.

[8] C. Bernon, M.-P. Gleizes, S. Peyruqueou, G. Picard, ADELFE: A Methodology for Adaptive Multi-agent Systems Engineering, in: Engineering Societies in the Agents World III. Springer Berlin / Heidelberg, 2003, pp 70-81.

[9] G. Beydoun, C. Gonzalez-Perez, B. Henderson-Sellers, G. C. Low, Developing and Evaluating a Generic Metamodel for MAS Work Products, in: A Garcia, R Choren, C Lucena, P Giorgini, T Holvoet, A Romanovsky (Eds) Software Engineering for Multi-Agent Systems IV: Research Issues and Practical Applications. Springer-Verlag, Berlin, 2006, pp 126-142.

[10] G. Beydoun, G. Low, B. Henderson-Sellers, H. Mouratidis, J. J. G. Sanz, J. Pavon, C. Gonzalez-Perez, FAML: a generic metamodel for MAS development, IEEE Transactions on Software Engineering 35(4) (2009).

[11] G. Beydoun, G. Low, H. Mouratidis, B. Henderson-Sellers, A security-aware metamodel for multi-agent systems (MAS), Information and Software Technology 51(5) (2009) 832-845.
[12] P. Bogg, G. Beydoun, G. Low, When to Use a Multi-Agent System?, The 11th Pacific Rim International Conference on Multi-Agents (PRIMA) (2008) 98-108.
[13] P. Bresciani , P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini, TROPOS:An Agent Oriented Software Development Methodology, Journal of Autonomous Agents and Multi-Agent Systems 8(3) (2004) 203-236.
[14] F. Buschmann, R. MEunier, H. Rohnert, P. Sommerland, M. Stahl, Pattern-Oriented Software Architecture Volume 1: A System of Patterns Wiley, Chichester, 1996.
[15] J. Castro, M. Kolp, J. Mylopoulos, Towards Requirements-Driven Information Systems Engineering: The Tropos Project, Information Systems (27) (2002) 365-389.
[16] R. Cattoni, A. Potrich, P. Charlton, E. Mamdani, Evaluating the FIPA standards on the field: an audio video entertainment applications, First Asia-Pacific Conference on Intelligent Agent Technology (1999).
[17] L. Cernuzzi, G. Rossi, On the Evaluation of Agent-Oriented Modeling Methods, Proceedings of the OOPSLA Workshop on Agent-Oriented Methodologies, 2002, pp.
[18] T. Chandarasupsang, S. Galloway, G. Burt, J. McDonald, T. Siewierski, Bidding behaviour and electricity market simulation, Probabilistic Methods Applied to Power Systems (2004) 349-355.
[19] L. Chen, K. Sycara, WebMate: A Personal Agent for Browsing and Searching, Proceedings of the 2nd International Conference on Autonomous Agents and Multi Agent Systems, Minneapolis, 1998.
[20] P.-C. Chu, An Object-Oriented Approach to Modeling Financial Accounting Systems, Accounting, Management and Information Technology 2(1) (1992) 39-56.
[21] M. Cossentino, From Requirements to Code with the PASSI Methodology, in: B Henderson-Sellers, P Giorgini (Eds) Agent-Oriented Methodologies. Idea Group, 2005, pp 79-106.
[22] M. Cossentino, N. Gaud, V. Hilaire, S. Galland, A. Koukam, ASPECS: an agent-oriented software process for engineering complex systems, Autonomous Agents and Multi-Agent Systems 20(2) (2010) 260-304.
[23] G. Cuní, M. Esteva, P. Garcia, E. Puertas, C. Sierra, T. Solchaga, MASFIT: Multi-Agent System for Fish Trading, Proceedings of the 16th European Conference on Artificial Intelligence (2004).
[24] K. Dam, M. Winkoff, Comparing Agent-Oriented Methodologies, in: P Giorgini, B Henderson-Sellers, M Winikiff. (Eds) Agent-Oriented Information Systems. Springer-Verlag, Berlin, 2004, pp 78-93.
[25] T. De Wolf, T. Holvoet, Towards a full life-cycle methodology for engineering decentralised multi-agent systems, Fourth International Workshop on Agent-Oriented Methodologies, San Diego, 2005.
[26] K. Decker, K. Sycara, A. Pannu, M. Williamson, Designing Behaviors for Information Agents, Proceedings of the First International Conference on Autonomous Agents, Marina del Rey, 1997.
[27] S. A. deLoach, M. Kumar, Multi-agent Systems Engineering: An Overview and Case Study, in: B Henderson-Sellers, P Giorgini (Eds) Agent-Oriented Methodologies. IDEA Group Publishing, 2005, pp 236-276.
[28] T. T. Do, M. Kolp, S. Faulkner, A. Pirotte, Introspecting Agent-Oriented Design Patterns in: SK Chang (Ed) Advances in Software Engineering and Knowledge Engineering. World Publishing, Singapore, 2004, pp 151-177.
[29] EURESCOM, MESSAGE: Methodology for engineering systems of software agents. Final guidelines for the identification of relevant problem areas where agent technology is appropriate, EURESCOM Project Report P907 (2001).
[30] F. J. Garijo, J. J. Gomez-Sanz, P. Massonet, The MESSAGE Methodology for Agent-Oriented Analysis and Design, in: B Henderson-Sellers, P Giorgini (Eds) Agent-Oriented Methodologies. IDEA Group Publishing, 2005, pp 203-235.
[31] M. Georgeff, A. Lansky, Reactive reasoning and planning: an experiment with a mobile robot, Proceedings of AAAI87, 1987.
[32] S. Giesecke, Taxonomy of architectural style usage, Proceedings of the 2006 conference on Pattern languages of programs, Portland, Oregon, 2006.
[33] G. Grozev, D. Batten, M. Anderson, G. Lewis, J. Mo, J. Katzfey, NEMSIM: Agent-based simulator for Australia's National Electricity Market, 8th Asia-Pacific Complex Systems Conference (Complex'07), Surfers Paradise, Australia, 2007.
[34] G. Grozev, D. Batten, M. Anderson, G. Lewis, J. Mo, J. Katzfey, NEMSIM: Agent-based simulator for Australia's National Electricity Market, SimTecT 2005 conference proceedings, Sydney, Australia, 2005.
[35] Z. Guessoum, L. Rejeb, R. Durand, Using adaptive Multi-Agent Systems to Simulate Economic Models, AAMAS04, New York, ACM, 2004, pp.
[36] J. F. Hair, R. Anderson, R. Tatham, W. C. Black, Multivariate data analysis, Prentice Hall, New York, 1992.
[37] A. L. G. Hayzelden, J. Bigham, Agent technology in communications systems: an overview, The Knowledge Engineering Review 14 (1999) 341-375.
[38] E. Horlait, Mobile Agents for Telecommunication Applications (Innovative Technology Series: Information Systems and Networks), in:Kogan Page Science, Portland, 2003, pp.
[39] N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, B. Odgers, J. L. Alty, Implementing a Business Process Management System using ADEPT: A Real-World Case Study, Int. Journal of Applied Artificial Intelligence 14(5) (2000) 421-465.
[40] K. Kida, K. Yoshifu, T. Asakura, T. Miyashita, Goodinator: Quick and Prudent Meeting Coordination with a Multi-Agent System, First Asia-Pacific Conference on Intelligent Agent Technology (1999).
[41] S. Lamparter, S. Becher, J.-G. Fischer, An Agent-based Market Platform for Smart Grids, Autonomous Agents and Multi Agent Systems Conference (AAMAS2010), Toronto, 2010.
[42] T. Lenox, S. Hahn, M. Lewis, T. Payne, K. Sycara, Task characteristics and intelligent aiding, Proceedings of the 2000 IEEE International Conference on Systems, Man, and Cybernetics (2000).
[43] V. R. Lesser, L. D. Erman, Distributed interpretation: a model and experiment, in: AH Bond, L Gasser (Eds) Distributed Artificial Intelligence. Morgan Kaufmann Publishers Inc., San Francisco, 1988, pp.
[44] Y. Liu, P. Massotte, P. Couturier, Self-Adaptation and Reconfiguration of an Agent-Based Production System: Virtual Factory, First Asia-Pacific Conference on Intelligent Agent Technology (1999).
[45] M. Ljungberg, A. Lucas, The OASIS air-traffic management system, Proceedings of the Second Pacific Rim International Conference on Artificial Intelligence (1992).
[46] E. Maston, S. DeLoach, An Organization-Based Adaptive Information System for Battlefield Situational Analysis, Integration of Knowledge Intensive Multi-Agent Systems (2003).

[47] B. Mayoh, T. Krink, E. G. Werk, D. Junping, How Flexible Should Agent Behaviour Languages Be?, First Asia-Pacific Conference on Intelligent Agent Technology (1999).
[48] N. Medvidovic, D. S. Rosenblum, Assessing the Suitability of a Standard Design Method for Modeling Software Architectures, Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1), 1999.
[49] M. Nodine, J. Fowler, T. Ksiezyk, B. Perry, M. Taylor, A. Unruh, Active Information Gathering in InfoSleuth, International Journal of Cooperative Information Systems 9(1/2) (2000) 3-28.
[50] I. Nourbakhsh, M. Lewis, K. Sycara, M. Koes, M. Yong, S. Burion, Human-Robot Teaming for Search and Rescue, IEEE Pervasive Computing (2005).
[51] S. A. O'Malley, S. DeLoach, Determining When to Use an Agent-Oriented Software Engineering Paradigm, in: Revised Papers and Invited Contributions from the Second International Workshop on Agent-Oriented Software Engineering II. Springer-Verlag, Berlin, 2002, pp 188-205.
[52] L. Padgham, M. Winikoff, Prometheus: A Practical Agent-Oriented Methodology, in: B Henderson-Sellers, P Giorgini (Eds) Agent-Oriented Methodologies. IDEA Group Publishing, 2005, pp 107-135.
[53] J. Pavon, J. Gomez-Sanz, Agent Oriented Software Engineering with INGENIAS, 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003), Prague, Czech Republic, 2003, pp. 394-403.
[54] M. Pěchouček, V. Mařík, Industrial deployment of multi-agent technologies: review and selected case studies, Autonomous Agents and Multi-Agent Systems 17(3) (2008) 397--431.
[55] M. Pěchouček, M. Rehák, V. Marik, Expectations and deployment of agent technology in manufacturing and defence: case studies, AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems (2005) 100-106.
[56] C. Reed, B. Boswell, R. Neville, Multi-agent Patient Representation in Primary Care, 10th Conference on Artificial Intelligence in Medicine 3581 (2005) 375-384.
[57] J. A. Rodriguez, On The Design and Construction of Agent-Mediated Electronic Institutions, Artificial Intelligence Research Insitute, UAB - Universitat Autonòma de Barcelona, 2003.
[58] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, New Jersey, 1995.
[59] A. Sabas, M. Badri, S. Delisle, A Multidimentional Framework for the Evaluation of Multiagent System Methodologies, Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI-2002), 2002, pp. 211-216.
[60] R. Schwartz, S. Kraus, Negotiation on data allocation in multi-agent environments, 14th National Conference on Artificial Intelligence, Providence, Rhode Island, MIT Presss, 1997, pp. 29-35.
[61] G. D. M. Serugendo, M.-P. Gleizes, A. Karageorgos, Self-Organisation in multi-agent systems, The Knowledge Engineering Review 20 (2005) 165-189.
[62] O. Shehory, A. Sturm, Evaluation of modeling techniques for agent-based systems, Proceedings of the fifth international conference on Autonomous agents, Montreal, 2001.
[63] C. J. Stanton, M. Williams, Grounding Robot Sensory and Symbolic Information Using the Semantic Web, Robocup 2003: Robot Soccer World Cup VII (2003).
[64] A. Sturm, O. Shehory, A framework for evaluating agent-oriented methodologies, in: AOIS 2003. Springer, 2004, pp 94-109.
[65] J. Sudeikat, L. Braubach, A. Pokahr, W. Lamersdorf, Evaluation of agent-oriented software methodologies - examination of the gap between modeling and platform, Agent-Oriented Software Engineering V (2004) 126--141.
[66] H. Sun, J. Thangarajah, L. Padgham, Eclipse-based Prometheus Design Tool, Proceedings of The 9th International Conference on Autonomous Agents and Multiagent Systems, Toronto, Canada, 2010, pp. 325-332.
[67] M. Tambe, W. L. Johnson, R. M. Jones, F. V. Koss, J. E. Laird, P. S. Rosenbloom, K. Schwamb, Intelligent Agents for Interactive Simulation Environments, AI Magazine (1995).
[68] Q. N. N. Tran, G. C. Low, Comparison of Ten Agent-Oriented Methodologies, in: B Henderson-Sellers, P Giorgini (Eds) Agent-Oriented Methodologies. Idea Group, Hershey, 2005, pp 341-367.
[69] Q. N. N. Tran, G. C. Low, MOBMAS: A Methodology for Ontology-Based Multi-Agent Systems Development, Information and Software Technology 50(7-8) (2008) 697-722.
[70] D. Walton, The New Dialectic, University of Toronto Press, Toronto, 1998.
[71] A. Weidlich, D. Veit, A critical survey of agent-based wholesale electricity market models, Energy Economics 30 (2008) 1728–1759.
[72] D. Weyns, A. Helleboogh, E. Steegmans, T. De Wolf, K. Mertens, N. Boucké, T. Holvoet, Agents are not part of the problem, agents can solve the problem, Proceedings of the OOPSLA Workshop on Agent-Oriented Methodologies (2004) 101-112.
[73] M. Wooldridge, An Introduction to Multi Agent Systems, Wiley, Chichester, 2002.
[74] M. Wooldridge, N. R. Jennings, F. Zambonelli, Multi-Agent Systems as Computational Organizations: The Gaia Methodology, in: B Henderson-Sellers, P Giorgini (Eds) Agent-Oriented Methodologies. IDEA Group Publishing, 2005, pp 136-171.
[75] E. S. K. Yu, Towards modelling and reasoning support for early-phase requirements engineering, IEEE International Symposium on Requirements Engineering, 1997, pp. 226-235.
[76] F. Zambonelli, N. Jennings, M. Wooldridge, Multi-Agent Systems as Computational Organizations: The Gaia Methodology, in: B Henderson-Sellers, P Giorgini (Eds) Agent-Oriented Methodologies. Idea Group, 2005, pp 136-171.
[77] F. Zambonelli, N. R. Jennings, M. Wooldridge, Developing Multiagent Systems: The Gaia Methodology, ACM Transactions on Software Engineering Methodology 12(3) (2003) 317-370.
[78] F. Zambonelli, A. Omicini, Challenges and research directions in agent-oriented software engineering, Autonomous Agents and Multi-Agent Systems 9(3) (2004) 253-283.

.

Tables

TABLE 1
FEATURES SUPPORTED BY RELATED WORK

Feature	Wooldridge [73]	Hayzelden [37]	Pěchouček [54, 55]	DeLoach [51]	De Wolf [25]	Eurescom [29]
Dynamic	Y	Y (some)		Y	Y	Y
Un certainty	Y					
Dependability						
Openness			Y			
Distributed Environment	Y		Y	Y	Y	
Communication Quality					Y	
Efficiency						
Robustness		Y	Y	Y	Y	
Reliability						
Flexibility		Y			Y	Y
Responsiveness		Y	Y			
Indeterminism		Y				Y
Concurrency						
Legacy	Y			Y		
Scalability		Y		Y		
Distributed Tasks	Y	Y	Y	Y		Y
Interaction Dependencies			Y	Y		Y
Interaction Type			Y	Y		Y

TABLE 2
DIFFERENCE IN RATINGS BY INDIVIDUAL FEATURES

Feature	Average difference across problem domains	Variance
Dynamic	0.167	0.038
Un certainty	0.433	0.038
Dependability	0.4	0.048
Openness	0.3	0.092
Distributed Environment	0.067	0.011
Communication Quality	0.267	0.043
Efficiency	0.2	0.048
Robustness	0.2	0.048
Reliability	0.133	0.027
Flexibility	0.367	0.055
Responsiveness	0.2	0.016
Indeterminism	0.267	0.059
Concurrency	0.033	0.007
Legacy	0.3	0.044
Scalability	0.1	0.028
Distributed Tasks	0.067	0.011
Interaction Dependencies	0.233	0.023
Interaction Type	0.067	0.011

TABLE 3
DIFFERENCE IN RATINGS ACROSS PROBLEM INSTANCES

Domain	Average difference across all features	Variance
Battlefield Information Systems	0.189	0.04
Intrusion Detection	0.244	0.05
Fish Auction (MASFIT)	0.255	0.079
Manufacturing	0.133	0.028
Disaster Robot Response	0.211	0.04
Document Recommendation	0.233	0.029

TABLE 4
COMPARISON TO AUTHOR'S RATINGS SET

Feature	Difference in averages between Author and Assessor A	Difference in averages between Author and Assessor B
Dynamic	0.333	0.233
Un certainty	0.167	0.467
Dependability	0.467	0.333
Openness	0.467	0.2
Distributed Environment	0.3	0.3
Communication Quality	0.133	0.133
Efficiency	0.267	0.4
Robustness	0.3	0.1
Reliability	0.133	0.133
Flexibility	0.267	0.367
Responsiveness	0.3	0.1
Indeterminism	0.133	0.267
Concurrency	0.2	0.233
Legacy	0.467	0.5
Scalability	0.267	0.333
Distributed Tasks	0.4	0.4
Interaction Dependencies	0.167	0.2
Interaction Type	0.267	0.267

TABLE 5
FACTOR ANALYSIS RESULTS

	Dynamic	Uncertain	Dependability	Openness	Distrib	Comm-Quality	Efficient	Robust	Reliable	Flexible	[redacted]	[redacted]	Concurren	Legacy	Scalable	Tasks	[redacted]	Type Interact
A	3	5	3	2	3	2	2	5	3	2	3	1	5	1	3	3	5	2
B	2	2	3	1	4	2	1	3	3	1	3	1	3	1	1	1	4	4
C	4	5	4	3	3	3	3	5	5	1	5	2	5	1	3	3	4	2
D	4	4	2	3	3	2	3	2	5	3	5	3	5	1	5	5	4	3
E	3	5	5	2	4	2	2	3	5	1	5	2	5	1	3	2	5	3
F	2	3	1	1	3	2	4	3	5	3	5	3	3	1	3	2	4	2
G	3	5	2	1	4	2	2	5	5	2	5	2	5	1	3	3	5	2
H	3	5	4	1	3	3	1	5	5	3	5	3	1	1	1	4	3	1
I	3	3	2	1	4	2	2	3	5	3	3	2	3	2	3	1	4	2
J	3	5	3	5	1	1	3	1	1	3	1	4	5	1	1	1	5	4
K	3	5	2	1	4	2	5	3	5	2	5	2	5	1	3	5	5	4
L	3	5	2	1	4	2	3	1	3	2	5	2	2	1	1	1	2	1
M	2	4	1	1	3	1	5	5	5	1	1	1	5	1	3	4	3	3
N	3	5	2	5	1	1	3	1	1	3	1	3	5	1	1	1	5	4
O	3	5	2	1	4	2	3	1	3	3	5	3	2	1	1	1	3	1
P	3	4	2	3	4	2	3	5	3	2	5	2	3	1	3	3	4	1
Q	3	3	1	1	3	2	2	2	5	1	3	2	2	1	3	4	5	3
R	4	5	3	5	2	2	3	5	5	2	5	3	5	1	3	5	5	4
S	2	4	2	1	4	2	3	5	5	2	5	2	5	1	5	5	4	2
T	2	2	2	3	2	1	2	3	3	1	5	3	5	1	1	4	4	4
U	4	5	3	4	2	3	2	5	3	1	5	2	5	1	2	3	5	2
V	3	5	3	1	2	1	3	2	3	3	5	4	1	1	1	1	2	1
W	2	3	1	1	2	1	4	5	5	1	3	3	3	1	3	3	4	2
X	2	3	1	1	4	2	1	1	3	3	3	3	1	1	1	1	2	1
Y	4	5	4	5	1	1	2	2	3	2	1	3	5	1	3	4	3	4

Ratings: 5 very high; 4 high; 3 moderate; 2 low; 1 very low)

A. Multi-agent patient care - [56]
B. MAS for conference management - [77]
C. Battlefield simulation - [46]
D. Intrusion detection - [4]
E. Masfit, fish auction MAS - [23]
F. Self-Adaptation and reconfiguration of an agent-based production system - [44]
G. WARREN – financial portfolio management MAS – [26]
H. Human-Robot Teaming Search and Rescue - [50]
I. Document recommendation tools - [53]
J. Simulating the evolution of societies - [47]
K. Business process management - [39]
L. InfoSleuth - [49]
M. EOSDIS - [60]
N. TacAir-Soar - [67]
O. RETSINA – WebMate - [19]
P. RETSINA – MokSAF - [42]
Q. AVEB - [16]
R. Robot Soccer - [63]
S. Aircraft controller - [45]
T. Meeting Scheduler - [40]
U. Distributed Sensing - [43]
V. Spacecraft Control - [31]
W. Manufacturing Pipeline - [77]
X. Personal Computer Management - [53]
Y. Robot Battles - [53]

TABLE 6
FACTOR ANALYSIS RESULTS

Feature	Factor #1	Factor #2	Factor #3	Factor #4	Factor #5
Dynamic			0.855		
Uncertainty			0.719		
Dependability			0.799		
Openness	0.721				
Distributed Environment	-0.698				
Communication Quality	-0.587				
Efficiency		0.574			
Robustness		0.679			
Reliability		0.749			
Flexibility				0.766	
Responsiveness	-0.666				
Indeterminism				0.796	
Concurrency	0.689				
Legacy					0.922
Scalability		0.861			
Distributed Tasks		0.795			
Interaction Dependencies	0.523				
Interaction Type	0.867				

TABLE 7
DESCRIPTIVE STATISTICS FOR THE RATINGS (SORTED BY MEAN)

Feature	Mean	Std. Deviation	Range
Uncertainty	4.2000	1.04083	2-5
Interaction dependencies	3.9600	1.01980	2-5
Responsiveness	3.8800	1.53623	1-5
Reliability	3.8800	1.30128	1-5
Concurrency	3.7600	1.53514	1-5
Robustness	3.1600	1.67531	1-5
Distributed environment	2.9600	1.05987	1-4
Dynamic	2.9200	.70238	2-4
Distributed tasks	2.8000	1.50000	1-5
Efficiency	2.6800	1.06927	1-5
Interaction-type	2.4800	1.15902	1-4
Indeterminism	2.4400	.82057	1-4
Scalability	2.4000	1.22474	1-5
Dependable	2.4000	1.08012	1-5
Openness	2.1600	1.54596	1-5
Flexibility	2.0400	.84063	1-3
Communication quality	1.8400	.62450	1-3
Legacy	1.0400	.20000	1-2

TABLE 8
GROUPINGS BY PEARSON'S CORRELATION AND FACTOR ANALYSIS

Group	Primary correlations (0.5 – 1.0)	Factor
1	Dynamic, Uncertain, (Un)dependable	3
2	Concurrency, Openness, Interaction Type, Interaction Dependencies.	1
3	Robustness, Reliability, Distributed Tasks, Scalability	2
4	Communication Quality, Responsiveness	1
5	Indeterminism, Flexibility	4
6	Efficiency	2
7	Legacy	5

TABLE 9
SUITABILITY SCHEME TABLE

Criteria & Features	**Rating** (Low <1-2, Moderate 3, High >4-5)
1. *Continual successful operation of software situated in dynamic, uncertain and undependable computing environments* *a. Dynamic* *b. Uncertainty* *c. Dependable*	_____ _____ _____
2. *The necessity for independent, concurrent software components to automate a process where inter-operations between new and existing components are required* *a. Openness* *b. Concurrency* *c. Interaction Dependencies* *d. Interaction Type*	_____ _____ _____ _____
3. *Continual successful operation of distributed software* *a. Distributed Tasks* *b. Robustness* *c. Reliability* *d. Scalability*	_____ _____ _____ _____
4. . *Successful operation of software where communication quality is expected to be low* *a. Communications Quality* *b. Responsiveness*	_____ _____
5. *The software solution is required to determine how to perform tasks, which tasks to perform, or integrate new tasks during its runtime* *a. Indeterminism* *b. Flexibility*	_____ _____

TABLE 10
SUITABILITY SCHEME TABLE FOR NEM APPLICATION BY TWO SOFTWARE DEVELOPERS, ONE OF WHICH IS A NON-AUTHOR.

Criteria & Features	**Rating** (Low <1-2, Moderate 3, High >4-5)	**Rating** (Low <1-2, Moderate 3, High >4-5)
1. *Continual successful operation of software situated in dynamic, uncertain and undependable computing environments*	5	4
a. Dynamic	3	3
b. Uncertainty	2	3
c. Dependable		
2. *The necessity for independent, concurrent software components to automate a process where inter-operations between new and existing components are required*	3	3
a. Openness	5	5
b. Concurrency	4	5
c. Interaction Dependencies	3	4
d. Interaction Type		
3. *Continual successful operation of distributed software*		
a. Distributed Tasks	3	4
b. Robustness	4	4
c. Reliability	4	4
d. Scalability	3	4
4. *Successful operation of software where communication quality is expected to be low*	2	2
a. Communications Quality	3	3
b. Responsiveness		
5. *The software solution is required to determine how to perform tasks, which tasks to perform, or integrate new tasks during its runtime*	*3*	*3*
a. Indeterminism	3	4

b. Flexibility		

TABLE 11

SUITABILITY SCHEME TABLE FOR FINANCIAL ACCOUNTING SYSTEM BY TWO SOFTWARE DEVELOPERS, ONE OF WHICH IS A NON-AUTHOR

Figures

Criteria & Features	Rating (Low <1-2, Moderate 3, High >4-5)	Rating (Low <1-2, Moderate 3, High >4-5)
6. *Continual successful operation of software situated in dynamic, uncertain and undependable computing environments*	1	1
d. Dynamic	1	1
e. Uncertainty	1	2
f. Dependable		
7. *The necessity for independent, concurrent software components to automate a process where inter-operations between new and existing components are required*	1	1
e. Openness	1	1
f. Concurrency	1	1
g. Interaction Dependencies	1	1
h. Interaction Type		
8. *Continual successful operation of distributed software*		
e. Distributed Tasks	1	1
f. Robustness	1	1
g. Reliability	1	2
h. Scalability	2	2
9. *Successful operation of software where communication quality is expected to be low*	1	1
c. Communications Quality	1	1
d. Responsiveness		
10. *The software solution is required to determine how to perform tasks, which tasks to perform, or integrate new tasks during its runtime*	*1*	*1*
c. Indeterminism	1	1
d. Flexibility		

Fig 1. NEM overview

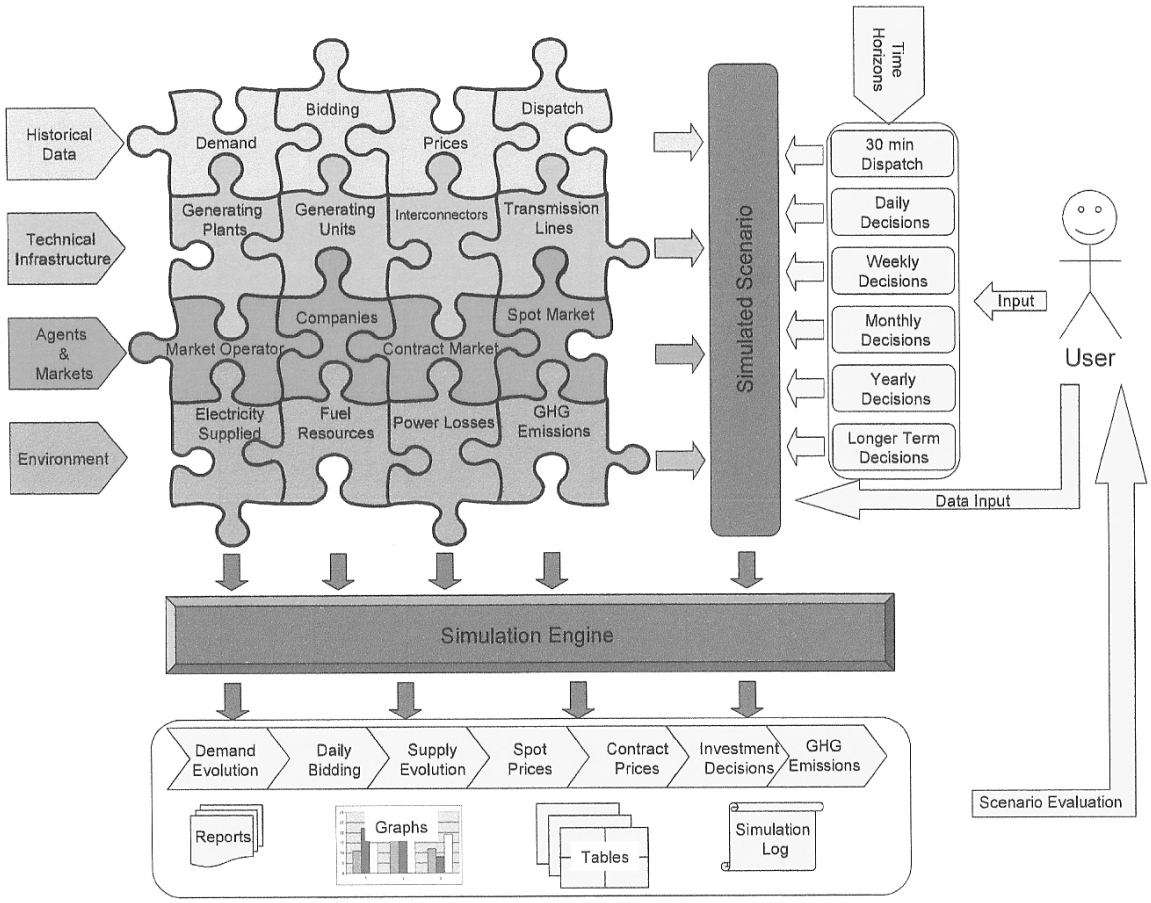

Fig 1. NEM overview structure[2] [33]

[2] Permission is granted for this material, presented at the 8th Asia-Pacific Complex Systems Conference (Complex'07), 2-5 July 2007, Surfers Paradise Marriott Resort, Queensland, to be available on the Complex'07 website to be shared for non-commercial, educational purposes, provided that this copyright statement appears on the reproduced material, and notice is given that the copying is by permission of the author(s).